SNLA LIBRARY
8446563

REFERENCE COPY
c.2

SANDIA REPORT

SAND91-8238 • UC-406
Unlimited Release
Printed January 1992

SAND91-8238
0002
UNCLASSIFIED

01/92
30P STAC

Overview of the DART Project

K. R. Berry, F. R. Hansen, L. M. Napolitano, E. L. McKelvey,
D. D. Andaleon, and J. E. Leeper

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94551
for the United States Department of Energy
under Contract DE-AC04-76DP00789

SF [illegible]-81

SAND91-8238
Unlimited Release
Printed January 1992

UC-406

Overview of the DART Project

K. R. Berry, F. R. Hansen, L.M. Napolitano,
E. L. McKelvey, D. D. Andaleon, and J. E. Leeper
Advanced Technology Department
Sandia National Laboratories
Livermore, CA 94551

Abstract

DART (DSP Arrary for Reconfigurable Tasks) is a parallel architecture of two high-performance SDP (digital signal processing) chips with the flexibility to handle a wide range of real-time applications. Each of the 32-bit floating-point DSP processors in DART is programmable in a high-level languate ("C" or Ada). We have added extensions to the real-time operating system used by DART in order to support parallel processors. The combination of high-level language programmability, a real-time operating system, and parallel processing support significantly reduces the development cost of application software for signal processing and control applications. We have demonstrated this capability by using DART to reconstruct images in the prototype VIP (Video Imaging Projectile) groundstation.

Introduction

Several programs at Sandia National Laboratories require real-time computational capabilities well in excess of what is readily available in conventional real-time computers. These efforts include intelligent control of manufacturing processes where measurements consist of real-time processing of video imagery, control of active structures (which can involve very high order plant models and controllers), and real-time automated target recognition (which again requires the ability to process images and extract information in real-time). All of these applications involve either very high speed data processing, very high speed data rates, or both. In the past, the computational requirements of such problems have been handled by custom designed hardware. Unfortunately, custom hardware provides an inflexible solution, and its expense can kill a project before it gets started.

With the recent introduction of floating-point DSP (digital signal processing) chips, a programmable solution to some of these high-end signal processing and control problems is now feasible. In 1990, the Advanced Technology Department at Sandia National Laboratories in Livermore, California began the DART (DSP Array for Reconfigurable Tasks) project to take advantage of the new DSP chips. Our goal was a system which could economically address some of these high-end signal processing and control applications. In order to address a range of applications, we chose a design using a parallel architecture which can be extended to provide the needed amount of computational power. To contain the cost of application software development, we wanted the system to be programmable in a high-level language with full operating system support. The DART concept is based on two Texas Instruments' TMS320C30 floating-point DSP chips. Texas Instruments provides several high level language compilers for the 'C30. DART uses a parallel architecture which can be extended by plugging DART boards together. It uses Spectron's SPOX real-time multi-tasking operating system with Sandia written extensions to support parallel processing.

This report describes the DART project, what we have done so far, and where we are going. Section 2 contains a hardware description of DART I, the first prototype DART board. Section 3 describes the extensions to the SPOX operating system we implemented to support parallel processing applications. Section 4 describes our first application of the DART processor, as the computational center of an artillery-fired close-range reconnaissance system. Finally, section 5 has some concluding remarks and a brief discussion of ongoing work.

DART I

The electronic hardware design portion of the DART I project involved designing an experimental board using a parallel architecture of DSP chips. This system allowed us to explore software and hardware issues involved in a parallel DSP system. Further, this system allowed a parallel DSP architecture to be used in solving real problems.

We designed a system with two digital signal processors for the original DART I system since this is the minimal set of processors required for parallel operation. An additional design consideration involved the host computer interface. The design team did not want to use valuable board space for specific host interface circuitry (e.g., interface to ISA bus or to VME bus). To simplify the host interface, we selected a PC-AT with a plug-in Banshee board (containing a single TMS320C30 DSP chip) from Atlanta Signal Processing Inc. (ASPI) as the host computer.

The basic architecture of the DART I system (see Figure 1) consists of two Texas Instruments' TMS320C30 DSP chips. The main memory (128K words of 32-bit memory) on the primary buses of the DSP chips is partitioned into two 64K word banks (memory banks 1 and 0). The two processors share a common primary bus with a bus switch between processors. This arrangement allows us to explore a common bus architecture and also a split bus architecture. A four-port memory (2K words x 32 bits accessible by any of four ports) connects the expansion ports of both 'C30s, the expansion port of the host computer and the primary memory bus on one side of the bus switch.

Processor Sites - Each processor site consists of a single TMS320C30 processor. An emulation header and interface circuitry provided for each 'C30 allows either or both processors to be connected to TI's XDS500 Emulator. Interconnected serial ports of both 'C30s provide serial communication between processors on the DART I system.

Memory Banks - Main memory on the DART I system consists of two 64K word (32-bit) SRAM memory banks. These memory banks are switch-settable to any 64K word address boundary within the TMS320C30's address space. The physical memory consists of 64K x 4 bit SRAMs. This configuration of SRAMs was a tradeoff between the depth of available high speed SRAMs and the amount of board space required for the memory chips.

Bus Switch - The bus switch consists of a collection of DIP switches that physically disconnects the electrical connection between the two buses. This feature allows the architectural configuration to be quickly

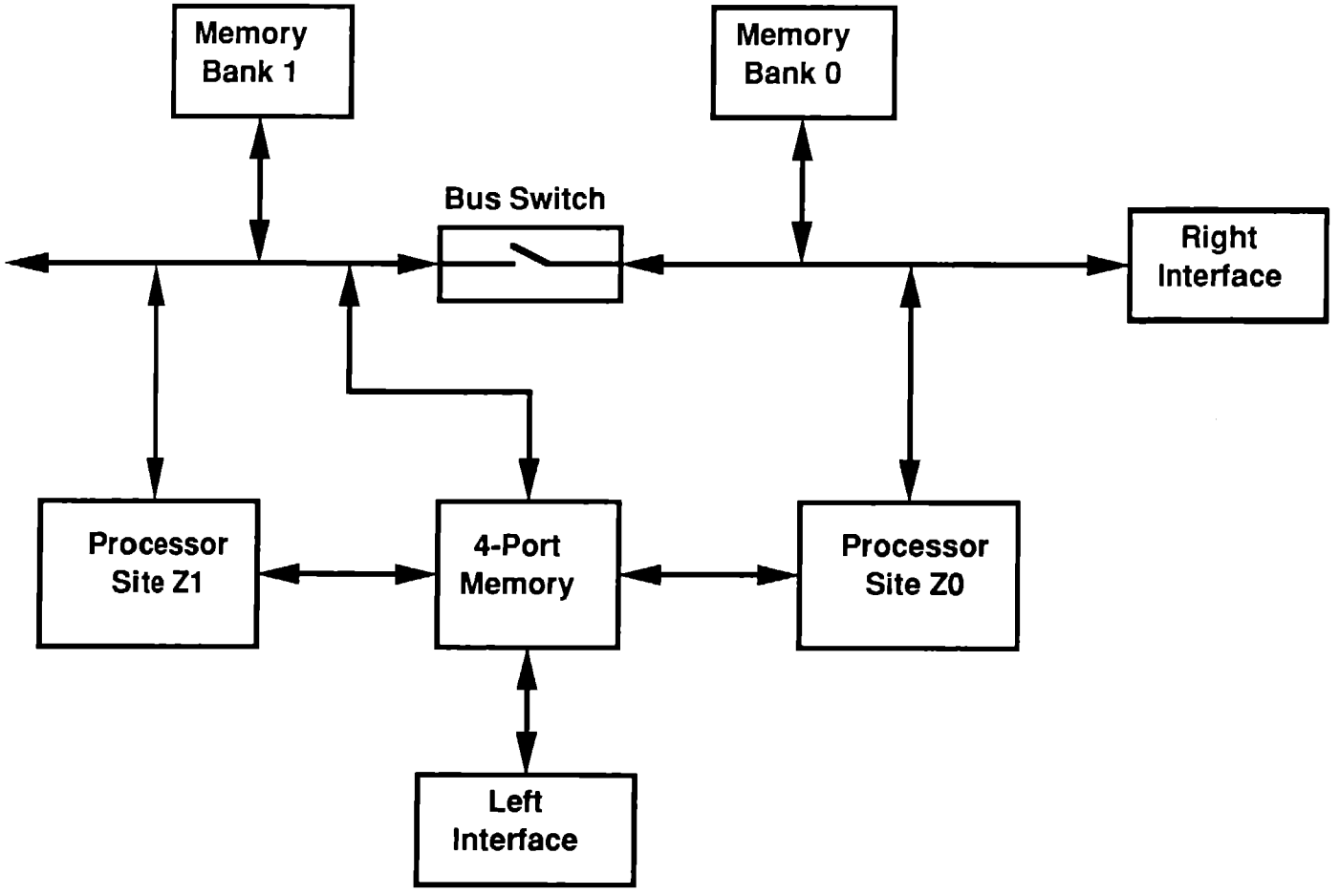


Figure 1
DART I Architecture

changed between a shared bus mode and a split bus mode. A bus arbiter managers the bus activity during shared bus mode operation.

4-Port Memory - The four-port memory is a full 32 bits wide and is 2K words deep. The expansion ports of both 'C30s connect to this common four-port memory. The two other ports of the four-port memory connect to the host computer and to the primary memory bus of one side of the bus switch. The common four-port memory allows the DART I processors to pass messages and data to each other. It also allows the processors to share data structures and instruction streams within this common memory space. Additionally, the four-port memory provides the primary communication path between the DART I system and the host computer. Finally, since the primary memory bus connection is memory-mapped into the boot space of the processors, the four-port memory serves as the boot memory for one or both processors (depending on the memory bus mode). Both expansion port connections are jumper selectable to reside in either the memory space or the I/O space of the 'C30.

Left Interface Connection - This interface consists of a direct connection from the physical connector to one of the ports of the four-port memory. This is the primary communication path between the DART I system and the host computer.

Right Interface Connection - This interface connects the primary memory bus of the DART I system to the host computer. It consists of bidirectional transceivers for the primary bus data lines and drivers for the primary bus address and control lines. This interface connection is switch-settable to any 64K word address boundary within the TMS320C30's address space.

The 'C30s can be controlled by the host computer. A software reset line from the host implements a common reset to the entire DART I system and places both 'C30s into reset. The host computer can also control the hold signals to each 'C30 independently. This allows the host computer to hold both processors, either one of the processors, or neither processor depending on the operation required.

The DART I board provides an external interface (consisting of both the left and right interfaces) patterned after ASPI's Banshee MPI (Multi-Processor Interface). The host Banshee has an MPI daughter card for connecting multiple Banshee cards. This interface uses multi-ported memory between processing elements (see Figure 2 for the logical block diagram). The Banshee/MPI unit consists of a single 'C30 and a dual-port memory. For the host computer to DART connection, the DART I board with two 'C30s is treated as a single processing element. The four-port memory on DART is

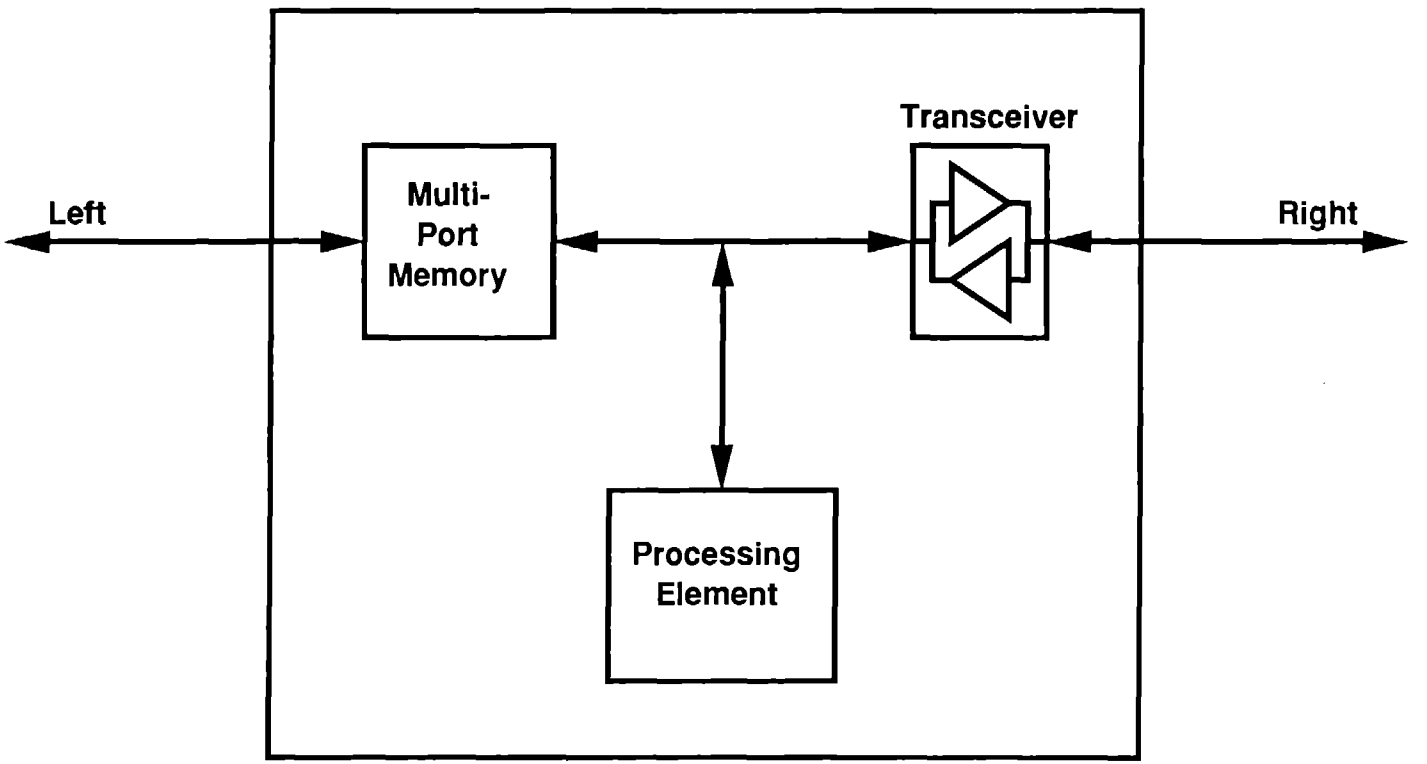


Figure 2
Simplified Multi-Processor Interface

used as the multi-ported memory between processing elements (*i.e.*, DART and Banshee). This allows either processor on the DART I board to communicate with the host through the shared memory of the four-port memory.

In addition to shared memory through the four-port memory, there are interrupt connections between the processors. Interrupts from all other processors are logically combined into a single interrupt line. Upon sensing an interrupt, the interrupted processor checks the communication register of each of the other processors for its own processor number. Once the interrupted processor finds its processor number, it knows which processor generated the interrupt and can begin servicing the request. This indirect scheme is expandable and is not limited by the number of interrupt lines into a processor, nor by the number of bit I/O lines out of a processor. A direct interrupt scheme for n processors requires $n-1$ interrupt lines into each processor and $n-1$ bit I/O lines out of each processor. The indirect interrupt scheme requires only one interrupt line into each processor and one bit I/O line out of each processor. However, additional external logic is required to logically combine the bit I/O lines from each processor into interrupt lines.

The boot sequence for the DART I system is controlled by the host computer. The host can control the reset and hold signals to the DART system. While the processors are in hold, the host computer places a loader routine into the boot space of the processor (4-port memory for shared memory mode; 4-port memory and 2-port memory of MPI board for split memory mode). Once the loader routine is in the place, the processor boot sequence is determined by the hold signals from the host computer. The loader routine loads program code from the host computer into DART's code space. The flexibility of this system allows researchers to make code changes and rapidly test the changes while running on the actual system. Also, the architecture can be switched between shared and split memory mode very quickly. The only hardware change on DART for mode changes is the settings of some DIP switches (a slightly modified loader routine must execute on the host computer).

The design of the DART I system provides a flexible architecture which allows us to rapidly experiment with different configurations. The two processor, parallel DSP system (with either split or shared primary memory) provides a useful prototype for developing solutions to real problems.

The DART I system and its host computer (PC-AT with Banshee card) can be viewed as a three DSP processor system. A three DSP processor system provides even more flexibility in the design of data flow architectures for various algorithmic applications. This type of multi-processor system provides an appropriate test bed for developing software extensions which support parallel processing in a multi-processor environment.

Software (Extensions to SPOX 1.3)

One of the initial goals of the DART project was to provide a high-level language programming environment for application programming. Therefore, we needed a real-time operating system which supports multi-tasking, parallel processing, and efficient high-speed I/O. We chose SPOX by Spectron Inc. It supports multi-tasking and provides architecture-independent, high-speed I/O through its device independent, streaming I/O calls. However, version 1.3 (the most recent when the project started) does not support a parallel processing environment. To meet our goals, we developed several extensions to SPOX to provide the needed parallel-processing support. This section will describe those extensions.

We set two requirements on the extensions. First, all the extensions that require inter-processor communication must be interrupt driven. This uses the least amount of processor overhead and ensures rapid response to a request between processors. Second, the parallel processor extensions should build upon the existing multi-tasking control concepts already in SPOX and should transparently provide control for multiple tasks on a single processor as well as tasks spread across several processors. From the programmer's point-of-view, an application which consists of several tasks running on several processors should be equivalent to running those same tasks on one processor. This approach accomplishes two things. First, the programmer need not worry about how to break the application up to run on multiple processors. He only needs to consider how to break it up to run as multiple interacting tasks. Second, it allows the code to be developed and debugged in a single processor environment. Converting the code to execute on multiple processor requires only changing I/O open statements to reflect remote machine names for inter-task communication.

These extensions consist of the following four modules. Each module supports a uni-processor as well as multi-processor environment:

1) The RS (Remote Signals) module provides remotely signaled conditions. It allows a task on one processor to wait for a condition to occur on another processor.

2) The NL (Network Locks) module provides network-wide resource locks.

3) The NB (Network Buffers) module provides network-wide buffers. This provides network-wide management of memory (including multi-ported memories). It is not an extension any existing SPOX concept.

4) The DTT (Driver, Task-to-Task communication device) module provides streaming communication channels between tasks running on the same or different processors. It is a SPOX compliant device driver that allows tasks to open channels to each other by task name and machine name.

RS (Remote Signals) Module.

The RS module contains one user-level function call, `RS_signal`. `RS_signal` provides the means for a task on one processor to wait on a condition (*e.g.*, a full data buffer or completion of a service request) to occur on a (potentially) different processor. It is based on the gates and conditions provided by the KG and KC modules of SPOX. A task enters a local gate created by the `KG_create` SPOX system call and waits on a condition created locally by the `KC_create` SPOX system call. This sleeping task can then be awakened by any other task on any other processor by the remote task issuing an `RS_signal` command. The only requirement is that the waiting task must have previously communicated with the signaling task to give it the addresses of the gate and condition it is waiting on. These remote signal calls are the primary task synchronization mechanism used to coordinate tasks on separate processors in DART. However, the RS module does not provide for network-wide gates. In SPOX, gates are used to protect shared data that can be accessed by several tasks. To provide this capability, we implemented network-wide resource locks described in the next subsection.

In addition to multi-processing support, we added another extension to SPOX's signals, the concept of a "return value". One common use of conditions is for tasks requesting services from one another. When a requesting task, "TASKA", requests a service of another task, "TASKB", that cannot be accomplished immediately (e.g., a calculation to be performed), TASKA goes to sleep on a previously agreed upon condition. When the service is finished, TASKB wakes up TASKA by signaling that condition. However, TASKA must now make another request to obtain the result. This is especially inefficient when the two tasks are on different processors and the request involves one processor interrupting the other. To avoid this second request, we added a return address to the condition object. This allows the servicing task (TASKB) to write (or, if the two tasks are on different processors, have the system write) a single word return value at that address for TASKA before it is awakened. This way, the second service request becomes unnecessary.

NL (Network Locks) Module

The NL module includes two user-level functions: `NL_seize` and `NL_release`. These two functions provide network-wide resource locks that mimic the uniprocessor resource locks provided in SPOX's IR module. The

NL module provides eight statically defined network locks: five are available for the user and three are reserved for the system. By calling `NL_seize`, a task seizes control of a particular lock and is assured exclusive use of the associated resource. Any other task on any other processor that tries to seize control of the same lock will block until the first task releases the lock with a `NL_release` call. When several tasks are requesting control of the same lock, it is handed out to each task on a first-come, first-served basis.

The services in the NL module are coordinated by one processor in the network designated as the NL manager. All requests for seizing and releasing locks are handled through interrupt service requests to that processor.

NB (Network Buffers) Module

The NB Module contains two user-level functions: `NB_seize` and `NB_release`. These functions coordinate the exclusive use of buffers within shared memory resources. DART I uses these functions to implement three buffers in the shared four-port memory. The DART I four-port memory is logically divided into four 512 word sections. The last section is reserved for system use. The other three buffers are available for use by applications.

The function `NB_seize` returns the address of the next free buffer and marks that buffer as used. No other `NB_seize` call on any other processor will return the same buffer until it is released with a `NB_release` call. As with the NL module, if all buffers are used, `NB_seize` will block until a buffer is released. If several tasks are waiting for a free buffer, the buffers are handed out to the waiting tasks on a first-come, first-served basis. Also as with the NL module, the NB services are coordinated by a single processor in the network designated as the NB manager. Requests to seize and release buffers are sent to this processor using interrupt service requests.

DTT (Driver, Task-to-Task communication device) Module

The DTT module provides streaming task-to-task communication channels. Using the inter-processor task control function provided by the extensions described above (i.e., RS,. NL, NB), we implemented streaming communication channels between tasks as a SPOX device driver. Thus, communication between tasks can be accomplished using standard SPOX streaming I/O calls (*e.g.*, `SS_open`, `SS_put`).

Tasks open communication channels to each other by processor name and task name. Communications are carried out by "putting" and "getting" frames of data into and out of the channels. For example, task "TASKA" on processor "DART0" can open a channel to task "TASKB" on processor "DART1" by simply issuing an `SS_open` call for the device "DART1:TASKB". Likewise TASKB must issue an `SS_open` call for the device

"DART0:TASKA". Once both tasks have issued corresponding open calls for each other, the channel is established. Each channel allows data movement in only one direction, so a pair of channels must be opened for bi-directional data movement. Communications require only `SS_put` and `SS_get` calls. We have demonstrated peak data transfer rates of 2.5M words/s (10M bytes/s) using the DTT device driver.

VIP Ground Station

In February 1988, a study was initiated at Sandia National Laboratories in Livermore, California to determine the feasibility of using a spin stabilized artillery projectile as a reconnaissance or imaging platform. The study was predicated on the perceived need for low-level military commanders to be able to obtain real-time combat information abut the area "over the next hill". The result of this study was VIP, the Video Imaging Projectile. This section will describe how DART's capability was exploited to prototype a ground station that can reconstruct video data telemetered down from the projectile and display it for viewing in real-time.

VIP background

A spin-stabilized projectile potentially provides a very low cost platform for obtaining close range imagery. The low cost potential is the result of the overall system simplicity. The concept uses a single optical sensor whose field-of-view is perpendicular to the longitudinal axis of the shell. As the shell spins and translates, the sensor scans a helical pattern whose axis is curved with the trajectory of the shell. The intersection of the helical pattern and the ground is a series of scan lines that forms rasters which can be reconstructed to form an image of the ground under the flight path (see Figure 3).

Two test units, VIP-1 and VIP-2, were designed and fabricated. They were based on the M549 155mm artillery shell shape and mass properties. Both test units were fired successfully in late 1989. All analog data telemetered back to the ground was recorded on high-speed 14-track analog tape to provide data for further laboratory experiments.

The resolution of the resulting image in the down range direction is a simple function of the shell's spin rate and forward velocity. For the 155mm howitzer, the shell travels forward approximately 3 meters per revolution at the gun barrel exit. As the shell flies, its forward velocity decreases faster than its rotational velocity and the range resolution improves to approximately 2 meters. The cross range resolution is a function of the projectile spin rate, the bandwidth of the telemetry link, the rate the analog signal is sampled, and the height of the projectile off the ground. For both VIP-1 and VIP-2, we sampled the analog signal at 1 MHz providing a ground cross-range resolution of approximately 1 meter when the shell is at 1 kilometer altitude.

The Problem: Image Reconstruction

The data stream to be processed, whether digitized directly from a telemetry link or from recorded tape, is essentially a continuous string of intensity values. To reconstruct an image of the ground over-flown by the

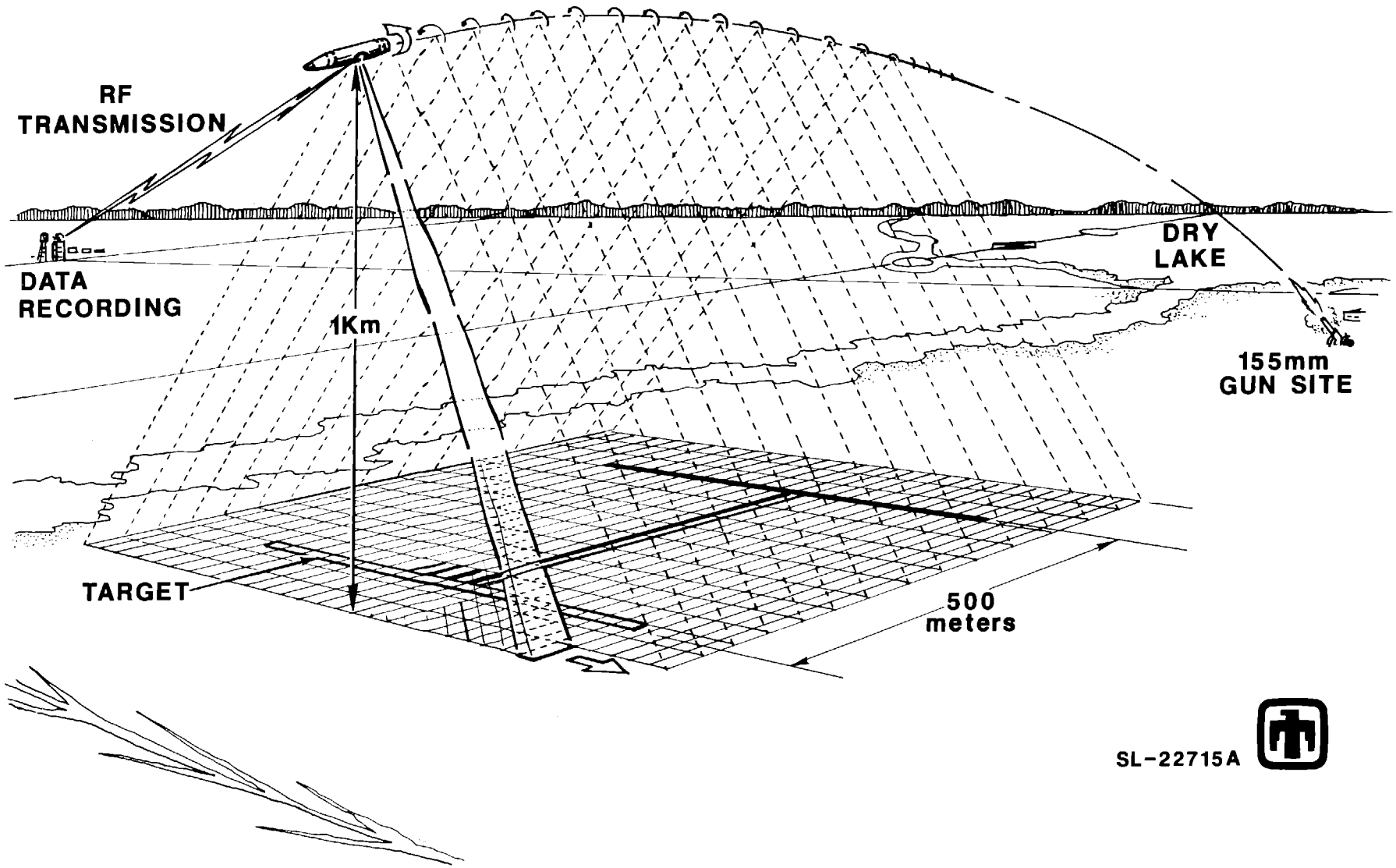


Figure 3
Video Imaging Projectile Test

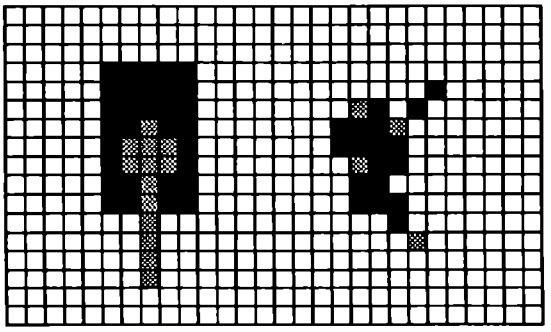
shell, this data stream must be sectioned into rasters. These rasters must then be registered one to the next. Alignment of the scan lines is automatic if a sufficiently accurate reference of the shell's angular orientation is available. For the current sampling rate of 1 MHz and the nominal spin rate for the 155mm shell of 150 Hz, each pixel in a scan line represents 1 milliradian. Figure 4 shows how sensitive the eye is to misregistration of adjacent lines, especially for small objects. In the figure, a few pixel misregistration makes it difficult to distinguish a tank-like vehicle from a bush. It was decided that the test objectives could be satisfied without a roll reference of 1 milliradian accuracy. Instead, the registration was accomplished using computational methods.

A gross reference was provided on the test shots by an infrared (IR) sensor mounted midway in the nose of the shell. The sensor saturates when the sun is in its field of view. The IR sensor reference is accurate only to ±10 milliradians. Therefore, to obtain acceptable image registration we must further computationally align the image. We do this by aligning each pair of adjacent raster lines so as to minimize the total absolute difference between them. VIP was originally conceived as an imaging platform which could provide field commanders with quick reconnaissance data of objects located over the next hill. To be effective in this role, the image must be reconstructed and displayed near the gun tube in near real-time. To demonstrate the feasibility of the system, a DART I based prototype groundstation was built that can digitize the video signal and then align the image for display in (near) real-time.

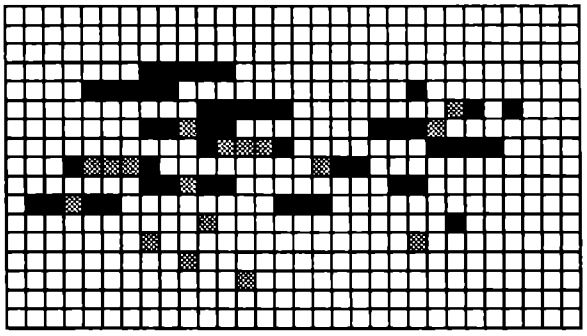
The Prototype Groundstation

A block diagram of the groundstation is shown in Figure 5. The groundstation uses DART I as its computational engine. A Compaq 386/25 serves as host to a Data Translation DT2862 frame grabber and an ASPI Banshee board containing one 'C30. The Banshee has 512K bytes of SRAM and 4M words of DRAM on a memory expansion daughter board. The Banshee serves as the system's control and communication processor and as the direct host to a DART I board. The video signal is digitized by a custom A/D board and the frame grabber drives the system display.

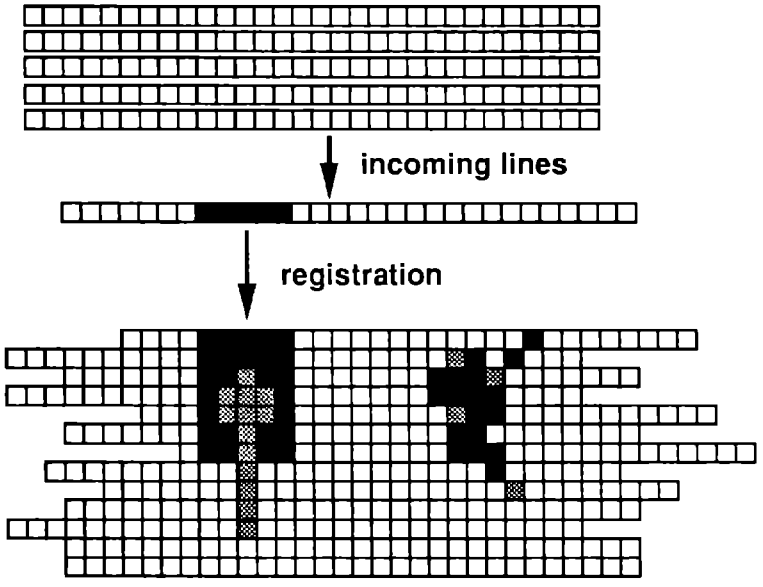
Using the parallel processor extensions described in Section 3, we implemented the software for image acquisition, registration, and display in a straightforward manner. The software consists of four cooperating tasks: one server task and three registration tasks. The server task runs on Banshee. It handles data acquisition, manages the registration process and displays the final image. The three registration tasks are identical and run on all three 'C30s, one per processor. They are responsible for registering continuous blocks of the image.



TRUE IMAGE



UNREGISTERED IMAGE
(AS RECEIVED IMAGE)



REGISTERED IMAGE

Figure 4: Effect of Image Registration

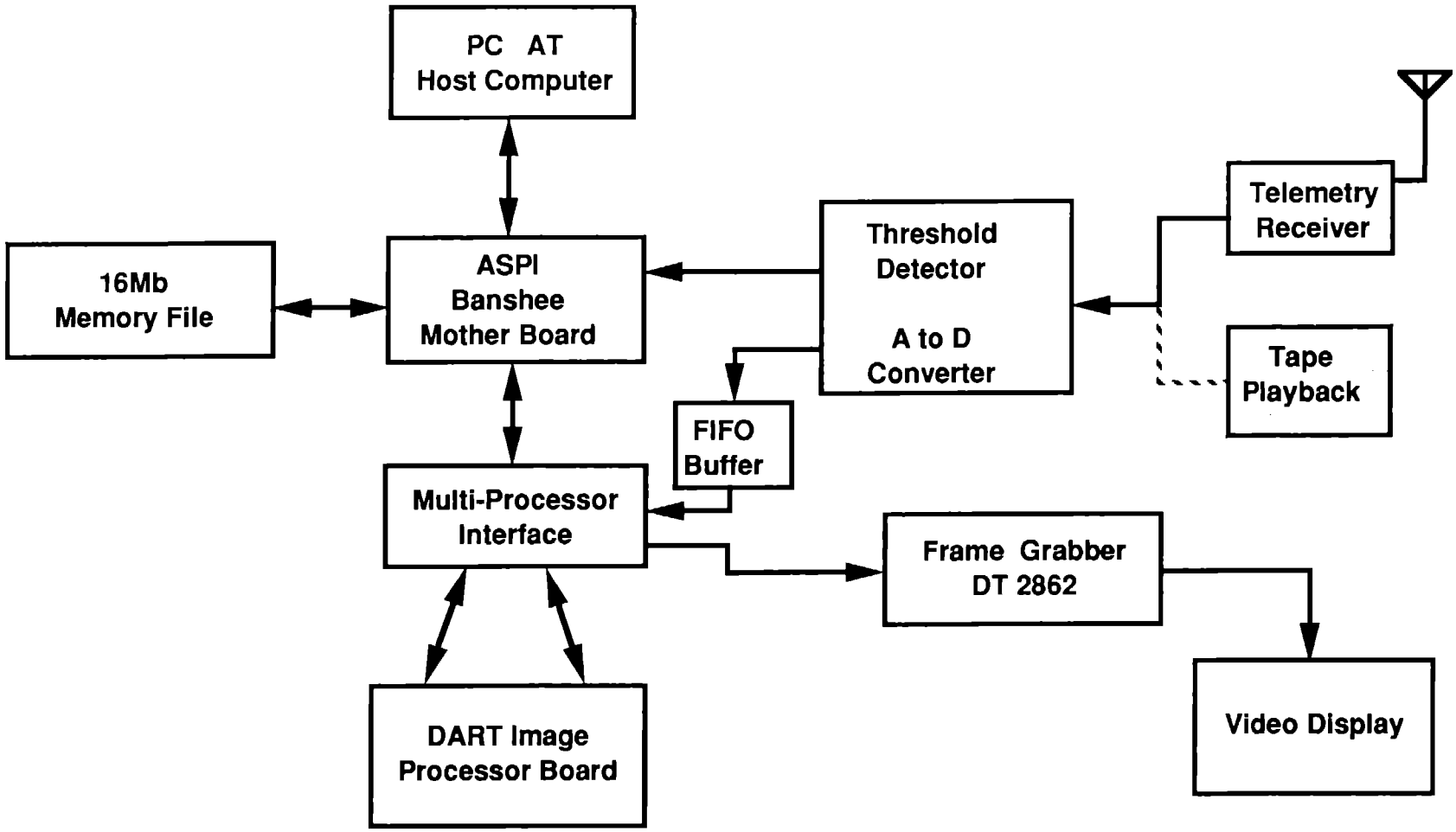
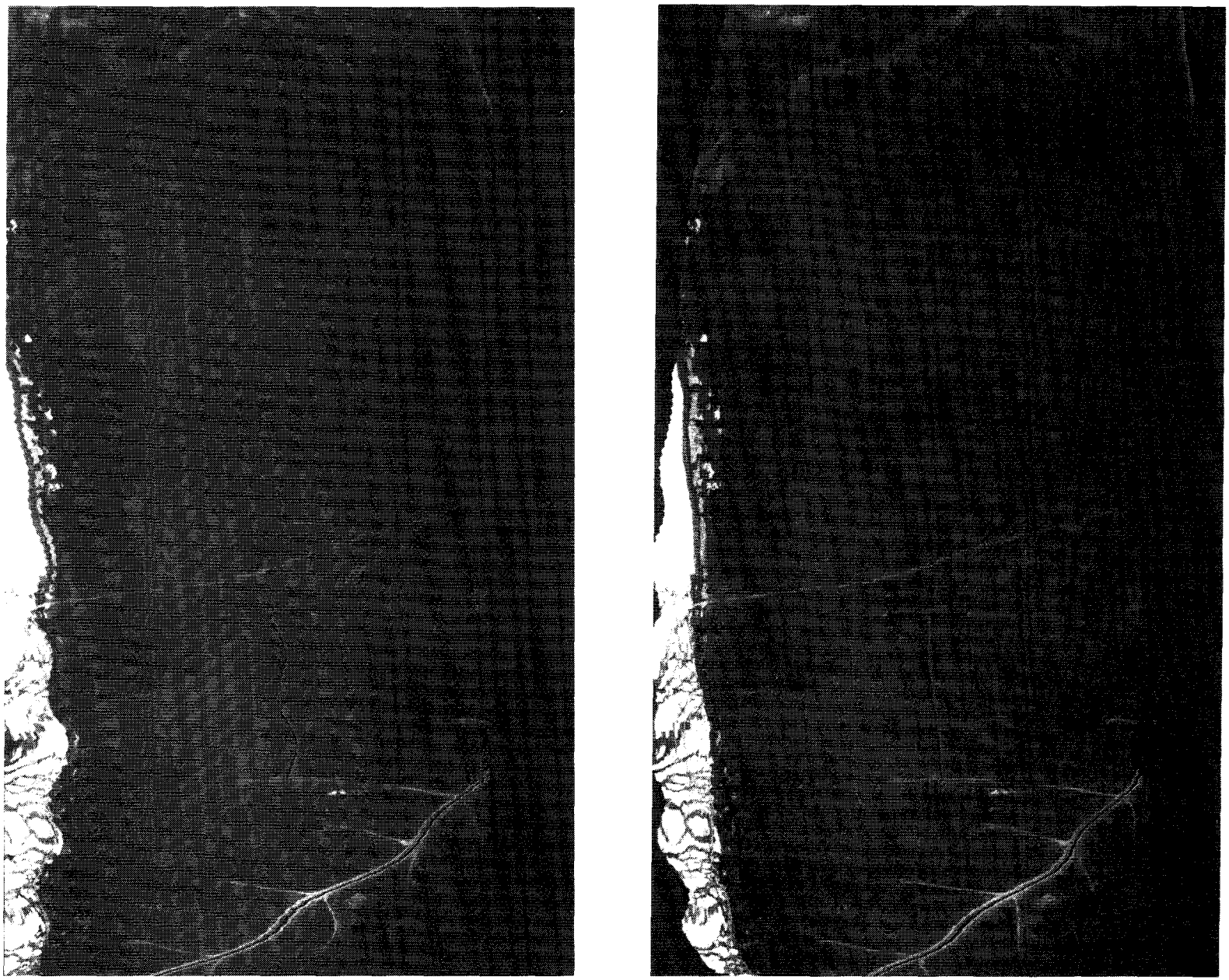


Figure 5
VIP Groundstation Prototype
Block Diagram

To acquire an image from a VIP shot, a controlling server task is loaded into Banshee and identical registration tasks are loaded into each 'C30 on DART I and on Banshee. First, using interrupts generated by the saturating IR signal as a time reference, the server task uses the high-speed A/D board to digitize that portion of the data corresponding to the ground (approximately one quarter of each revolution) and stores consecutive raster lines in the DRAM daughter board. Due to the limited size of the DRAM board, only 12 seconds of a 30 second VIP flight are stored. This corresponds to 1500 consecutive rasters, each consisting of 1500 pixels.

Once the digitized image has been stored, the image is aligned in a second pass. The server passes the image to the registration tasks via the DTT communication channels, in blocks of eight consecutive raster lines. The registration tasks then independently align consecutive pairs of rasters within their blocks and pass the relative raster-to-raster offsets back to the server task, again via DTT channels. The server task repeats this process, passing out blocks of rasters to each registration task and collecting back relative offsets, until the entire image has been registered. The server task then displays the registered image on a monitor via the frame grabber. The server also provides mouse driven pan and zoom for detailed image inspection.

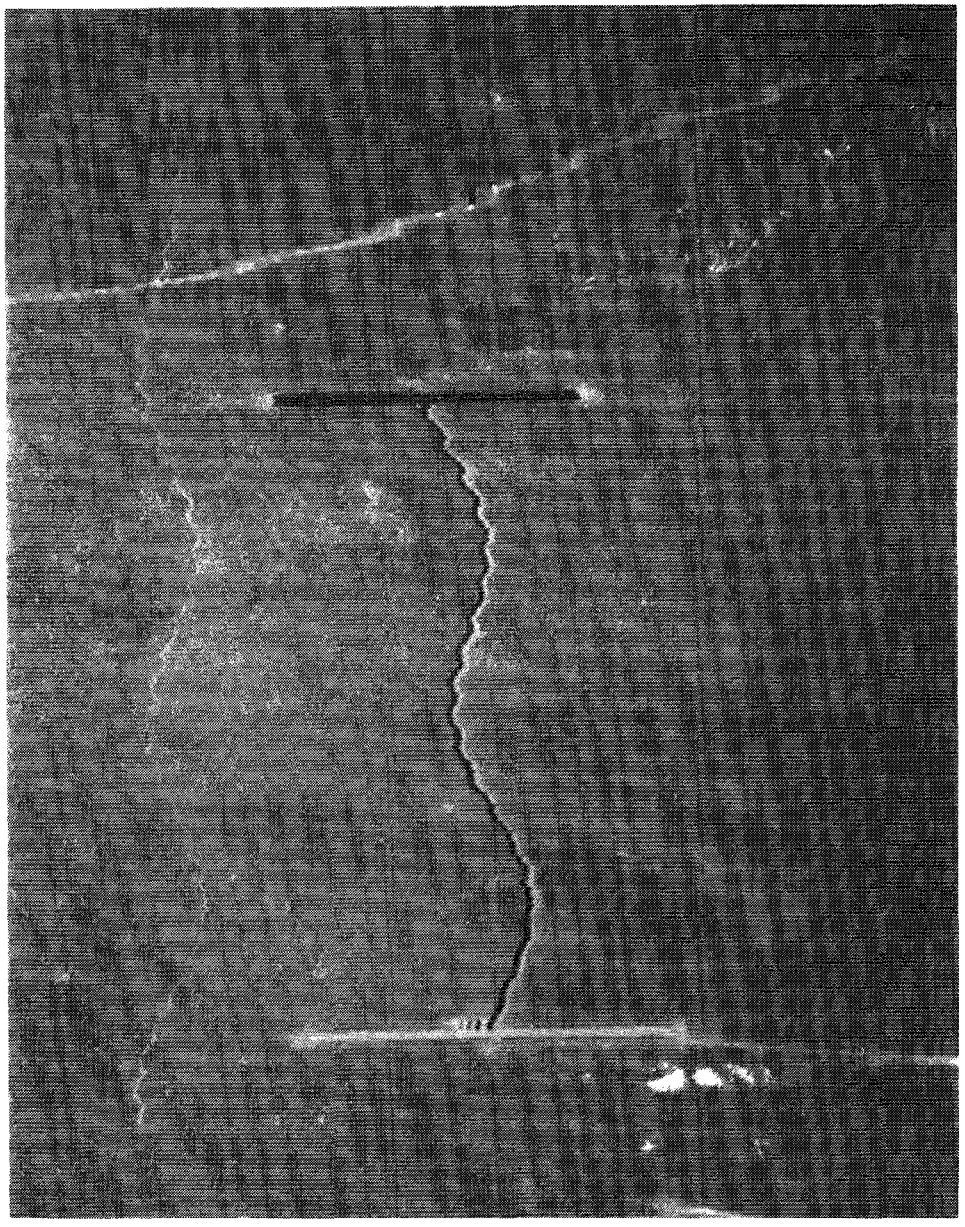
Both the VIP-1 and VIP-2 shots were conducted at Sandia's Tonopah Test Range in Nevada. Figure 6 shows a side by side comparison of an unregistered versus a registered image of 12 seconds of flight from the VIP-2 shot. The "waviness" in the unregistered image is due to the effect of the shell's coning dynamics on the IR sensor's field-of-view. Although distortions have not been completely eliminated from the registered image, it is significantly better than the unregistered image. The entire image is registered in just over 13 seconds. In actual use, the center 12 seconds of data can be aligned and displayed on the screen before the shell hits the ground. Figure 7 shows a side by side closeup comparison of the unregistered versus registered "I" shaped test pattern. The "I" shaped pattern was painted on the ground prior to the shot to test the imaging quality of VIP. It is 500 meters long and 250 meters wide. The lines are 4 meters wide.



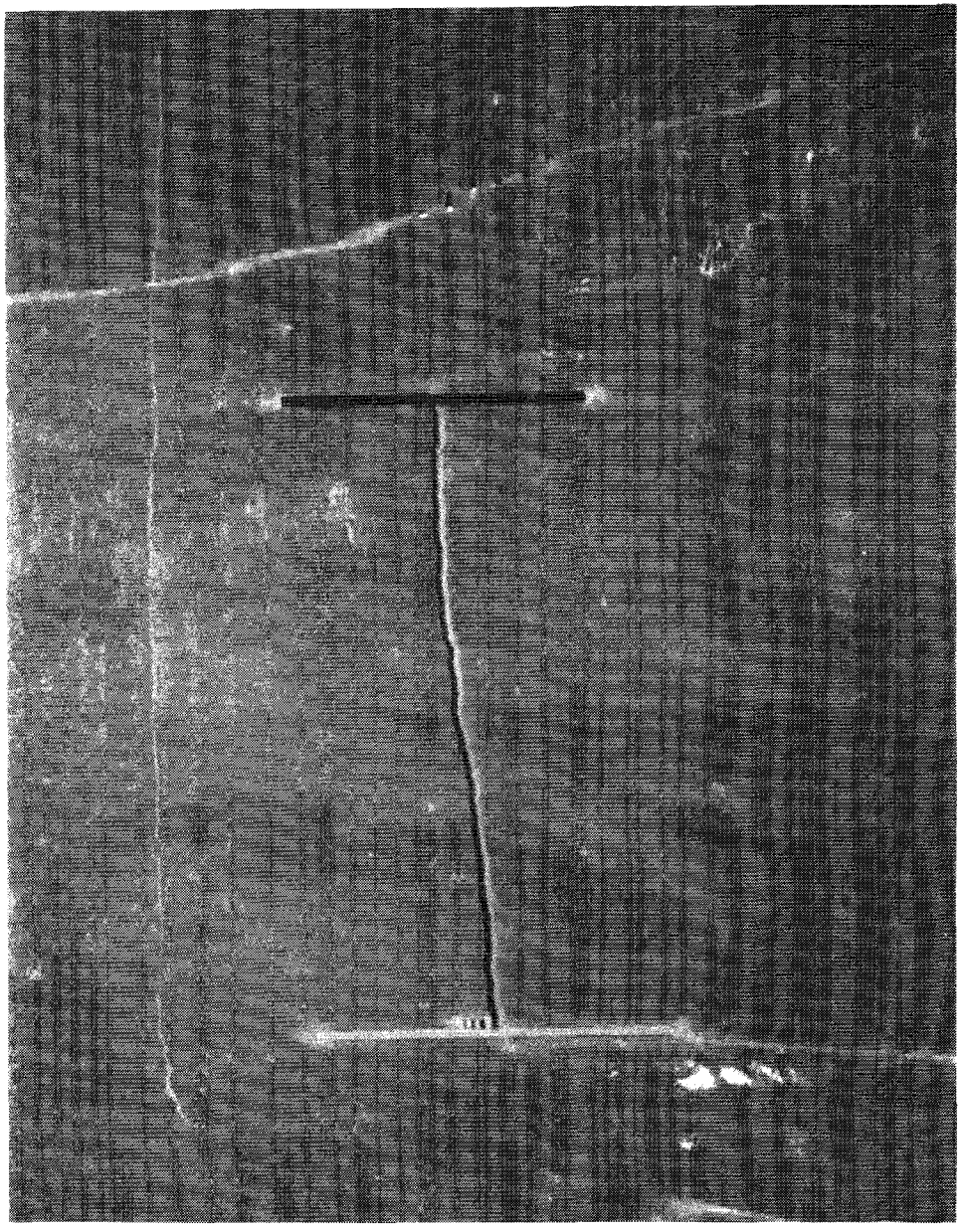
Unregistered Image

Registered Image

Figure 6: VIP Flight Data (12 sec.)



Unregistered Image



Registered Image

Figure 7: I-shaped Test Pattern

Conclusion & Future Work

High-end signal processing and control applications require significant computational capabilities. The DART I board (with two 'C30s) can perform up to 66 MFLOPs in a parallel processing environment. The DART I board is programmable in a high-level language ("C" or Ada) and uses the SPOX real-time multi-tasking operating system;. We have added our own local extensions to support parallel processing: RS (Remote Signals), NL (Network Locks), NB (Network Buffers), and DTT (Driver, Task-to-Task communication device). This combination of software features ("C" programmability, the SPOX operating system, and parallel processing support) significantly reduces the development cost of application software for signal processing and control applications. We have successfully applied this capability to a real-world, real-time problem (image reconstruction for VIP), thus demonstrating the viability of using parallel DSPs for high-end signal processing problems.

One of the lessons learned from the experimental applications of the DART I system was that even a 66 MFLOP DSP board cannot keep up with certain applications requiring high I/O throughput or very high floating-point performance. Luckily, many of these applications do have algorithms that can be partitioned into parallel or serial structures. Thus, we are designing a next generation DART board that can be a more flexible, modular building block for connecting into a multi-board system with either parallel or serial communication strategies.

The DART II design improves on the original DART I design and make DART II a more practical solution for existing problems (*e.g.*, intelligent control, structural control, and image processing). The design of DART II improves on the original dual-processor design by: 1) providing for multiple DARTs to be connected in series or in parallel, 2) incorporating bi-directional transceivers between the primary buses of both processors, 3) increasing the amount of available memory, and 4) packaging the electronics onto a standard PC-AT plug-in card.

The basic architecture of the DART II system is shown in Figure 8. The system consists of two Texas Instruments' TMS320C30 DSP chips. A dual-port memory (8K words x 32 bits) connects the expansion ports of both 'C30s. The main memory (256K words of 32-bit memory) on the primary buses of the DSP chips is partitioned into two 128K word "semi-private" banks. A single set of address and data transceivers across the entire primary bus allows the two processors to "share" their semi-private memory banks. This arrangement eliminates arbitration overhead for private memory accesses. In addition, on the primary memory bus of one of the processors is a large, field-programmable gate array attached to all data, address and control lines. This gate array serves as a reprogrammable interface to either the outside world or

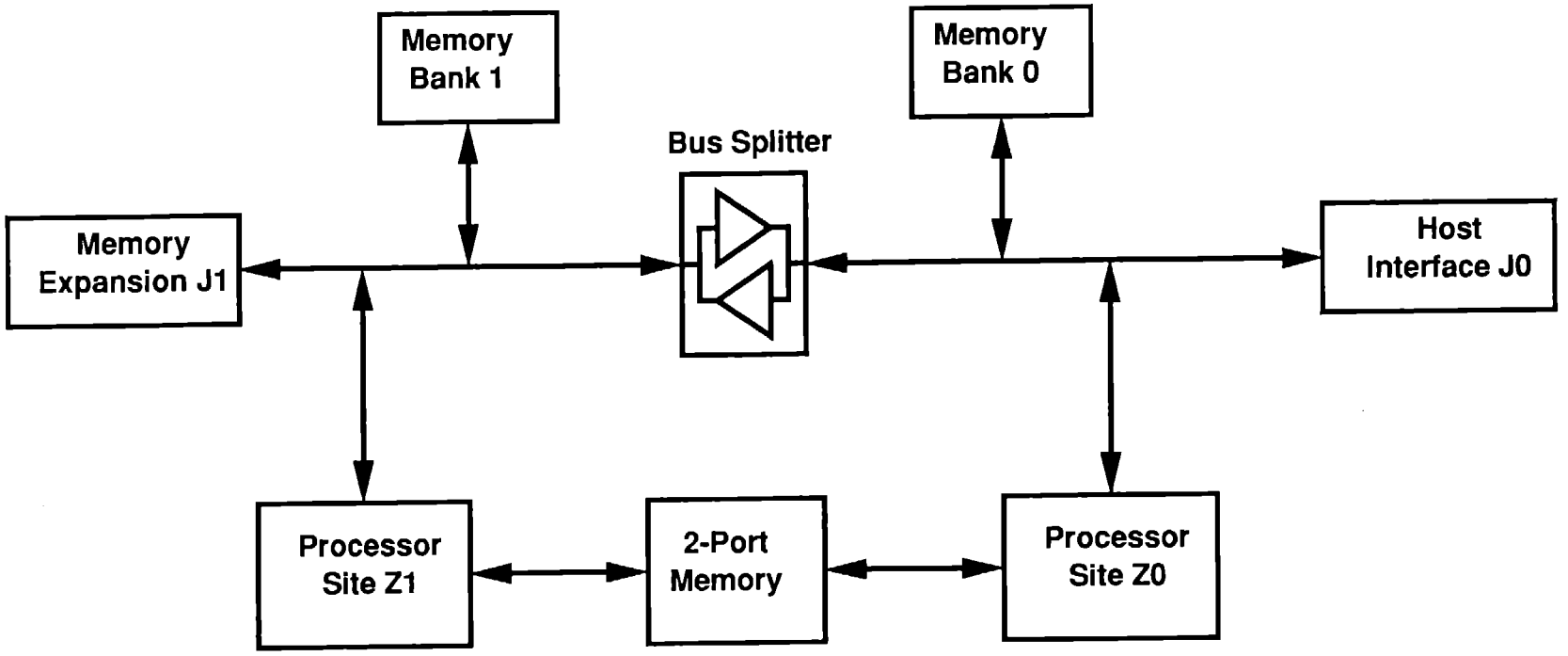


Figure 8
DART II Architecture

to other DART II cards. This interface allows multiple DART II cards to be connected in series or in parallel, whichever is best suited for a particular application (see figure 9).

The design of the DART II system provides a flexible, expandable architecture which can be used to solve practical problems. The DART II module allows a system designer to quickly design an electronic system architecture that is tailored to address his specific problem, and with the available compiler and operating system tools, he can produce quick turnaround, turnkey systems for a variety of uses.

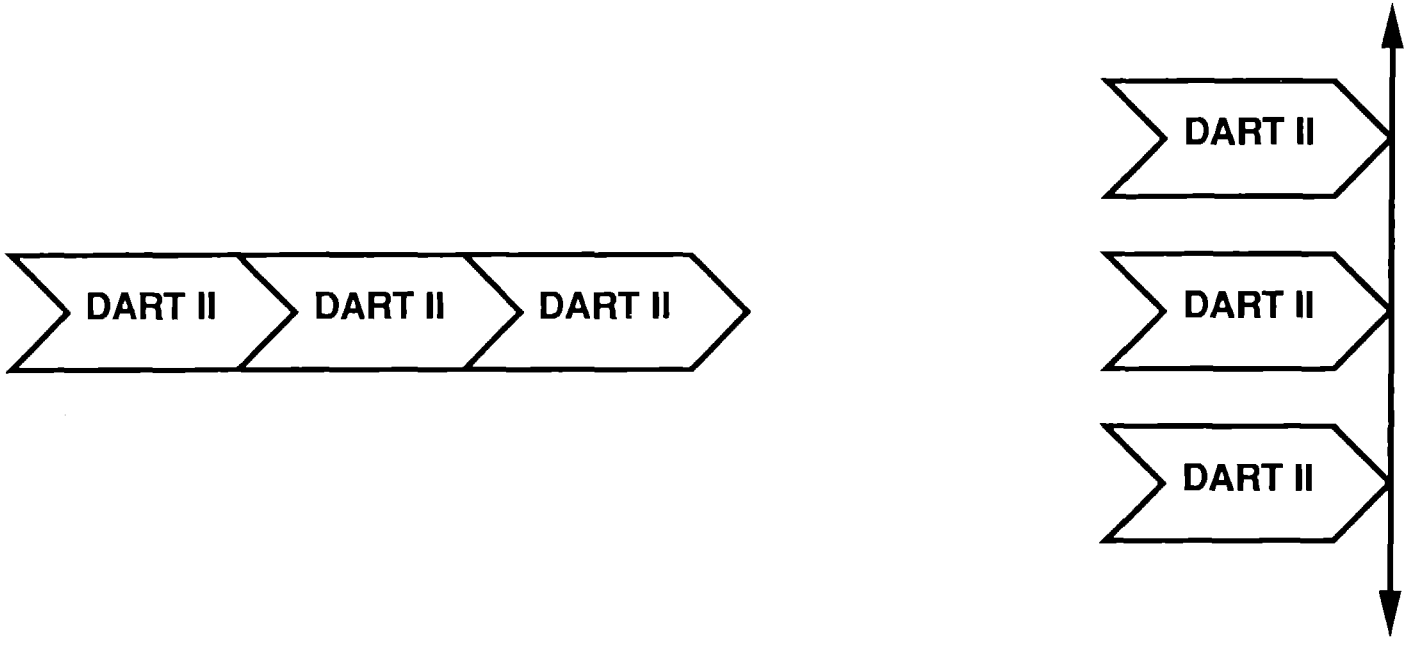


Figure 9
Connection Strategies

UNLIMITED RELEASE

INITIAL DISTRIBUTION

1128 P. J. Hargis, Jr.
1164 T. D. Raymond
1400 E. H. Barsis
1900 D. L. Crawford
2234 S. M. Kohler
2336 H. Shen
2345 B. L. Burns
2722 M. S. Rogers
2732 P. A. Mahoney
5375 D. D. Andaleon
5375 C. T. Oien
8000 J. C. Crawford
Attn: E. E. Ives, 5300
R. J. Detry, 8200
P. L. Mattern, 8300
P. E. Brewer, 8500
8245 R. J. Kee
8300A T. M. Dyer
8300A J. Vitko
8362 R. W. Carling
8400 R. C. Wayne
8430 L. A. Hiles
8431 F. R. Hansen (20)
8432 K. R. Berry (20)
8432 J. S. Kraabel
8432 J. E. Leeper
8432 E. L. McKelvey
8432 L. M. Napolitano (20)
8432 W. G. Wilson
8433 M. H. Rogers
8440 H. Hanser
8450 L. A. Hiles
Attn: T. R. Harrison, 8451
C. L. Knapp, 8453
A. L. Hull, 8454
8454 M. T. Stewart
8455 C. F. Acken
8480 L. A. West
8484 L. N. Tallerico
9130 A. C. Watts
9131 D. D. Boozer
9133 L. D. Hostetler

9543 J. C. Matter
9561 C. J. Hanley
9567 S. C. Roehrig

8535 Publications for OSTI (10)
8535 Publications/Technical Library Processes Division, 3141
3141 Technical Library Processes Division (3)
8524-2 Central Technical Files (3)